```
ebugger.foil
```

```
- ---------------------------------------------------------------------
-          debugger logic presentation
- ---------------------------------------------------------------------

ignal bpsta_we_l              : std_logic; -- BD

ignal bpl_we_l                : std_logic_vector( 4 downto 1); -- C7, C4, C1, BE
ignal bph_we_l                : std_logic_vector( 4 downto 1); -- C8, C5, C2, BF
ignal bnk_we_l                : std_logic_vector( 4 downto 1); -- C9, C6, C3, C0

ignal jtml_we_l               : std_logic; -- CA
ignal jtmh_we_l               : std_logic; -- CB

ignal sbk_we_l                : std_logic; -- CE
ignal bpcrl_we_l              : std_logic; -- CF .

ignal scratch_pad_we_l : std_logic_vector(15 downto 1); -- D1 -> DF

- ---------------------------------------------------------------------
- ESFR registers
-
- BPSTA: Break Point Status register address: hBD
- ---------------------------------------------------------------------

ignal bpsta_sb_reg_l     : std_logic;  -- BPSTA reg SB (stack trap)
ignal bpsta_b_reg_l      : std_logic_vector(4 downto 1);  -- BPSTA reg B4-B1

ignal debugger_active    : std_logic;  -- break point condition occur, in debugger mode and it
                                       -- should temporarily disable all break point function
ignal bpsta_resume       : boolean;    -- when write 80h to BPSTA, write-protect ESFR[BE-CF]

ignal reg_unprotect_l    : std_logic;  -- enable write to ESFR[BE-CF]
ignal pc_update_l        : std_logic;  -- enable the PCH/PCL update

- ---------------------------------------------------------------------
- Break point registers, (LB, HB,BANK) 1-4
- ---------------------------------------------------------------------

ignal bpl1               : std_logic_vector(7 downto 0);  -- Break point reg-1 (LB)
ignal bpl2               : std_logic_vector(7 downto 0);  -- Break point reg-2 (LB)
ignal bpl3               : std_logic_vector(7 downto 0);  -- Break point reg-3 (LB)
ignal bpl4               : std_logic_vector(7 downto 0);  -- Break point reg-4 (LB)

ignal bph1               : std_logic_vector(7 downto 0);  -- Break point reg-1 (HB)
ignal bph2               : std_logic_vector(7 downto 0);  -- Break point reg-2 (HB)
ignal bph3               : std_logic_vector(7 downto 0);  -- Break point reg-3 (HB)
ignal bph4               : std_logic_vector(7 downto 0);  -- Break point reg-4 (HB)

ignal bnk1_ben           : std_logic;                       -- Break point bank reg-1 (BEN)
ignal bnk2_ben           : std_logic;                       -- Break point bank reg-2 (BEN)
ignal bnk3_ben           : std_logic;                       -- Break point bank reg-3 (BEN)
ignal bnk4_ben           : std_logic;                       -- Break point bank reg-4 (BEN)

ignal bnk1_b             : std_logic_vector(3 downto 0);  -- Break point bank reg-1 (B[3:0])
ignal bnk2_b             : std_logic_vector(3 downto 0);  -- Break point bank reg-2 (B[3:0])
ignal bnk3_b             : std_logic_vector(3 downto 0);  -- Break point bank reg-3 (B[3:0])
ignal bnk4_b             : std_logic_vector(3 downto 0);  -- Break point bank reg-4 (B[3:0])

ignal bp1_adr            : std_logic_vector(15 downto 0); -- Break point 1 address
ignal bp2_adr            : std_logic_vector(15 downto 0); -- Break point 2 address
ignal bp3_adr            : std_logic_vector(15 downto 0); -- Break point 3 address
ignal bp4_adr            : std_logic_vector(15 downto 0); -- Break point 4 address

  ---------------------------------------------------------------------
  Jump to Monitor address LB, HB, stack break point register adr: CA, CB, CE
  ---------------------------------------------------------------------

gnal jtml_l              : std_logic_vector(7 downto 0);
gnal jtmh_l              : std_logic_vector(7 downto 0);
gnal sbk_l               : std_logic_vector(7 downto 0);

  ---------------------------------------------------------------------
  Break point control register        adr: CF
  ---------------------------------------------------------------------

  bpcrl_be_reg_l -> BE4, BE3, BE2, BE1 address break point enable
  bpcrl_s_reg_l  -> Stack trap condition
  bpcrl_bpe_reg_l -> global debugger enable
  ---------------------------------------------------------------------

gnal bpcrl_bpe_reg_l     : std_logic;
gnal bpcrl_s_reg_l       : std_logic_vector( 1 downto 0);
gnal bpcrl_be_reg_l      : std_logic_vector( 4 downto 1);  -- for BE4, BE3, BE2, BE1

  ---------------------------------------------------------------------
  PC copy register (LB, HB)        adr: F9 - FA
  ---------------------------------------------------------------------

gnal pcl_l               : std_logic_vector(7 downto 0);
gnal pch_l               : std_logic_vector(7 downto 0);

  ---------------------------------------------------------------------
  debugger circuitry signals for stack trap and address break point trap
  ---------------------------------------------------------------------
```

```
signal stack_trap_l         : std_logic;    -- detect stack trap when global debugger is enabled
signal stack_trap_dl        : std_logic;    -- latched stack trap
signal stack_trap_d2        : std_logic;
signal stack_trap_d3        : std_logic;
signal stack_trap_d4        : std_logic;
signal stack_trap_d5        : std_logic;

signal stack_trap_jump_l: std_logic;

signal bp1_trap_l           : std_logic;
signal bp2_trap_l           : std_logic;
signal bp3_trap_l           : std_logic;
signal bp4_trap_l           : std_logic;

signal bp1_trap_active      : std_logic;    -- bp1_trap_l and (not bp1_trap_toggle_l)
signal bp2_trap_active      : std_logic;
signal bp3_trap_active      : std_logic;
signal bp4_trap_active      : std_logic;


-- -----------------------------------------------------------------------------
-- When first detect address break point condition, bpx_toggle_l is 0. After return from
-- Monitor program, the return address match break point address and bpx_toggle_l is 1, so
-- it won't trigger address break point condition to avoid this infinitive loop.
-- -----------------------------------------------------------------------------

signal bp1_toggle_l         : std_logic;
signal bp2_toggle_l         : std_logic;
signal bp3_toggle_l         : std_logic;
signal bp4_toggle_l         : std_logic;

signal jump_ready           : std_logic;    -- ready to jump to monitor program
-- signal jump_l             : std_logic;
signal jump_adr_sel         : std_logic_vector(1 downto 0);   -- select 02H, jtmh_., jtml_l

signal jump_proga_en        : std_logic;    -- jump_proga_en <= prog_mux_control_l and Proga_en_i
signal clear_jump_l         : boolean;
signal prog_mux_control_l: std_logic;       -- prog_mux_control_l = 1, select jump address
                                            -- prog_mux_control_l = 0, normal operation


-- -----------------------------------------------------------------------------
-- external SFR register block
-- -----------------------------------------------------------------------------

-- -----------------------------------------------------------------------------
-- BPSTA: Break Point Status Register   -- address hBD
--
-- B4, B3, B2, B1 -> break point address X trigger break condition
-- SB -> Stack Trap trigger break condition
--
-- When break condition occur, it will set one of B4, B3, B2, B1, SB.
--
-- When write h80 to this register it will clear reg. to 0, write-protect ESFR[BE-CF]
-- When write h55 to this register it will unprotect ESFR(BE-CF).
-- -----------------------------------------------------------------------------

bpsta_resume <= (bpsta_we_l = '1' and Destin_do_i(7 downto 0) = "10000000");   -- write 80h to BPSTA

bpsta_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then   -- asynchronous reset
    bpsta_b_reg_l     <= "0000";
    bpsta_sb_reg_l    <= '0';

  elsif (Clk_i = '1' and Clk_i'event) then

    if (bpsta_resume) then   -- synchronous reset
      bpsta_b_reg_l     <= "0000";
      bpsta_sb_reg_l    <= '0';
    else
      if (bp1_trap_active = '1') then
        bpsta_b_reg_l(1)    <= '1';
      end if;

      if (bp2_trap_active = '1') then
        bpsta_b_reg_l(2)    <= '1';
      end if;

      if (bp3_trap_active = '1') then
        bpsta_b_reg_l(3)    <= '1';
      end if;

      if (bp4_trap_active = '1') then
        bpsta_b_reg_l(4)    <= '1';
      end if;

      if (stack_trap_jump_l = '1') then
        bpsta_sb_reg_l    <= '1';
      end if;

    end if;
  end if;
```

```
-------------------------------------------------------------------
debugger_active -> break point condition occurs, it should temporarily disable break point
                   trigger function.
-------------------------------------------------------------------
```

```vhdl
bugger_active <= bpsta_b_reg_1(1) or bpsta_b_reg_1(2) or bpsta_b_reg_1(3) or bpsta_b_reg_1(4) or
                 bpsta_sb_reg_1;
```

```
-------------------------------------------------------------------
reg_unprotect_1 = 0 -> ESFR[BE-CF] are write protected.
reg_unprotect_1 = 1 -> ESFR[BE-CF] are unprotected.

When jump to Monitor, automatically unprotect.
-------------------------------------------------------------------
```

```vhdl
otect_reg: process (Clk_i, Resetz_exclude_USB_i)
gin
  if (Resetz_exclude_USB_i = '0') then
     reg_unprotect_1  <=  '0'; -- reset to write-protected after power on

  elsif (Clk_i = '1' and Clk_i'event) then
     if ( (bpsta_we_1 = '1' and Destin_do_i(7 downto 0) = "01010101") or -- write 55H
          (prog_mux_control_1 = '1')   -- Jump to Monitor
        )then
        reg_unprotect_1  <=  '1'; -- write unprotected when write 55H
     elsif (bpsta_resume) then
        reg_unprotect_1  <=  '0'; -- write-protected when write 80H
     end if;
  end if;
d process protect_reg;
```

```
-------------------------------------------------------------------
Write h80 to BPSTA: Break Point Status register to enable
PCL/PCH update.

When jump is ready, it disable PCL/PCH update.
-------------------------------------------------------------------
```

```vhdl
_update_reg: process (Clk_i, Resetz_exclude_USB_i)
gin
  if (Resetz_exclude_USB_i = '0') then
     pc_update_1  <=  '0';

  elsif (Clk_i = '1' and Clk_i'event) then
     if (bpsta_resume) then
        pc_update_1  <=  '1'; -- Write 80H to BPSTA enable PCH/PCL update
     elsif (jump_ready = '1' ) then   -- break point address is stored in PCL/PCH
        pc_update_1  <=  '0'; -- disable the update when reach breakpoint
     end if;
  end if;
d process pc_update_reg;
```

```
-------------------------------------------------------------------
Break point register-1 (LB)   address - hex BE
-------------------------------------------------------------------
```

```vhdl
ll_reg: process (Clk_i, Resetz_exclude_USB_i)
gin
 if (Resetz_exclude_USB_i = '0') then
   bpll    <= "00000000"; -- power-up default value
 elsif (Clk_i = '1' and Clk_i'event) then

   if ((bpl_we_1(1) = '1') and (reg_unprotect_1 = '1')) then
      bpll <=  Destin_do_i(7 downto 0);
   end if;
 end if;
d process bpll_reg;
```

```
-------------------------------------------------------------------
Break point register-1 (HB)   address - hex BF
-------------------------------------------------------------------
```

```vhdl
ph1_reg: process (Clk_i, Resetz_exclude_USB_i)
gin
 if (Resetz_exclude_USB_i = '0') then
   bph1    <= "00000000"; -- power-up default value
 elsif (Clk_i = '1' and Clk_i'event) then

   if ((bph_we_1(1) = '1') and (reg_unprotect_1 = '1')) then
      bph1 <=  Destin_do_i(7 downto 0);
   end if;
 end if;
d process bph1_reg;
```

```
-------------------------------------------------------------------
Break point bank register-1   address - hex C0
-------------------------------------------------------------------
```

```vhdl
nk1_reg: process (Clk_i, Resetz_exclude_USB_i)
gin
 if (Resetz_exclude_USB_i = '0') then
   bnk1_b   <= "0000";     -- power-up default value
   bnk1_ben <= '0';
```

```vhdl
    elsif (Clk_i = '1' and Clk_i'event) then

      if ((bnk_we_l(1) = '1') and (reg_unprotect_l = '1')) then
        bnk1_b   <=  Destin_do_i(3 downto 0);  -- break point address - bank value
        bnk1_ben <=  Destin_do_i(7);  -- bank value comparison enable
      end if;
    end if;
end process bnk1_reg;

-- --------------------------------------------------------------------
-- Break point register-2 (LB)   address - hex C1
-- --------------------------------------------------------------------

bpl2_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bpl2    <= "00000000";  -- power-up default value
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bpl_we_l(2) = '1') and (reg_unprotect_l = '1')) then
      bpl2 <=  Destin_do_i(7 downto 0);
    end if;
  end if;
end process bpl2_reg;

-- --------------------------------------------------------------------
-- Break point register-2 (HB)   address - hex C2
-- --------------------------------------------------------------------

bph2_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bph2    <= "00000000";  -- power-up default value
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bph_we_l(2) = '1') and (reg_unprotect_l = '1')) then
      bph2 <=  Destin_do_i(7 downto 0);
    end if;
  end if;
end process bph2_reg;

-- --------------------------------------------------------------------
-- Break point bank register-2   address - hex C3
-- --------------------------------------------------------------------

bnk2_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bnk2_b   <= "0000";       -- power-up default value
    bnk2_ben <= '0';
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bnk_we_l(2) = '1') and (reg_unprotect_l = '1')) then
      bnk2_b   <=  Destin_do_i(3 downto 0);  -- break point address - bank value
      bnk2_ben <=  Destin_do_i(7);  -- bank value comparison enable
    end if;
  end if;
end process bnk2_reg;

-- --------------------------------------------------------------------
-- Break point register-3 (LB)   address - hex C4
-- --------------------------------------------------------------------

bpl3_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bpl3    <= "00000000";  -- power-up default value
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bpl_we_l(3) = '1') and (reg_unprotect_l = '1')) then
      bpl3 <=  Destin_do_i(7 downto 0);
    end if;
  end if;
end process bpl3_reg;

-- --------------------------------------------------------------------
-- Break point register-3 (HB)   address - hex C5
-- --------------------------------------------------------------------

bph3_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
    bph3    <= "00000000";  -- power-up default value
  elsif (Clk_i = '1' and Clk_i'event) then

    if ((bph_we_l(3) = '1') and (reg_unprotect_l = '1')) then
      bph3 <=  Destin_do_i(7 downto 0);
    end if;
  end if;
end process bph3_reg;

-- --------------------------------------------------------------------
-- Break point bank register-3   address - hex C6
-- --------------------------------------------------------------------
```

```vhdl
bnk3_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
   if (Resetz_exclude_USB_i = '0') then
      bnk3_b   <= "0000";      -- power-up default value
      bnk3_ben <= '0';
   elsif (Clk_i = '1' and Clk_i'event) then

      if ((bnk_we_1(3) = '1') and (reg_unprotect_1 = '1')) then
         bnk3_b   <= Destin_do_i(3 downto 0);  -- break point address - bank value
         bnk3_ben <= Destin_do_i(7);  -- bank value comparison enable
      end if;
   end if;
end process bnk3_reg;

-- ------------------------------------------------------------------
-- Break point register-4 (LB)   address - hex C7
-- ------------------------------------------------------------------

bpl4_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
   if (Resetz_exclude_USB_i = '0') then
      bpl4     <= "00000000";  -- power-up default value
   elsif (Clk_i = '1' and Clk_i'event) then

      if ((bpl_we_1(4) = '1') and (reg_unprotect_1 = '1')) then
         bpl4 <= Destin_do_i(7 downto 0);
      end if;
   end if;
end process bpl4_reg;

-- ------------------------------------------------------------------
-- Break point register-4 (HB)   address - hex C8
-- ------------------------------------------------------------------

bph4_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
   if (Resetz_exclude_USB_i = '0') then
      bph4     <= "00000000";  -- power-up default value
   elsif (Clk_i = '1' and Clk_i'event) then

      if ((bph_we_1(4) = '1') and (reg_unprotect_1 = '1')) then
         bph4 <= Destin_do_i(7 downto 0);
      end if;
   end if;
end process bph4_reg;

-- ------------------------------------------------------------------
-- Break point bank register-4   address - hex C9
-- ------------------------------------------------------------------

bnk4_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
   if (Resetz_exclude_USB_i = '0') then
      bnk4_b   <= "0000";      -- power-up default value
      bnk4_ben <= '0';
   elsif (Clk_i = '1' and Clk_i'event) then

      if ((bnk_we_1(4) = '1') and (reg_unprotect_1 = '1')) then
         bnk4_b   <= Destin_do_i(3 downto 0);  -- break point address - bank value
         bnk4_ben <= Destin_do_i(7);  -- bank value comparison enable
      end if;
   end if;
end process bnk4_reg;

-- ------------------------------------------------------------------
-- Jump to Monitor adr reg (LB) address - hex CA
-- ------------------------------------------------------------------

jtml_regs: process (Clk_i, Resetz_exclude_USB_i)
begin
   if (Resetz_exclude_USB_i = '0') then
      jtml_1   <= "00000000";

   elsif (Clk_i = '1' and Clk_i'event) then

      if (jtml_we_1 = '1' and reg_unprotect_1 = '1') then
         jtml_1 <= Destin_do_i(7 downto 0);
      end if;
   end if;
end process jtml_regs;

-- ------------------------------------------------------------------
-- Jump to Monitor adr reg (HB) address - hex CB
-- ------------------------------------------------------------------

jtmh_regs: process (Clk_i, Resetz_exclude_USB_i)
begin
   if (Resetz_exclude_USB_i = '0') then
      jtmh_1   <= "00000000";

   elsif (Clk_i = '1' and Clk_i'event) then

      if (jtmh_we_1 = '1' and reg_unprotect_1 = '1') then
```

```
debugger.foil

        jtmh_l <=  Destin_do_i(7 downto 0);
    end if;
  end if;
end process jtmh_regs;

-- ---------------------------------------------------------------
-- Stack break point register    address - hex CE
-- ---------------------------------------------------------------

sbk_regs: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
     sbk_l    <= "00000000";

   elsif (Clk_i = '1' and Clk_i'event) then

     if (sbk_we_l = '1' and reg_unprotect_l = '1') then
       sbk_l <=  Destin_do_i(7 downto 0);
     end if;
   end if;
end process sbk_regs;

-- ---------------------------------------------------------------
-- Break point control register address - hex CF
-- ---------------------------------------------------------------

bpcrl_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
     bpcrl_bpe_reg_l   <=   '0';
     bpcrl_s_reg_l     <=  "00";
     bpcrl_be_reg_l    <=  "0000";

   elsif (Clk_i = '1' and Clk_i'event) then

     if (bpcrl_we_l = '1' and reg_unprotect_l = '1') then
        bpcrl_bpe_reg_l   <=  Destin_do_i(7);
        bpcrl_s_reg_l     <=  Destin_do_i(5 downto 4);
        bpcrl_be_reg_l    <=  Destin_do_i(3 downto 0);
     end if;
   end if;
end process bpcrl_reg;

-- ---------------------------------------------------------------
-- PC copy register (LB, HB)         address - hex F9, FA
-- ---------------------------------------------------------------

pc_reg: process (Clk_i, Resetz_exclude_USB_i)
begin
  if (Resetz_exclude_USB_i = '0') then
     pcl_l   <= (others => '0') ;
     pch_l   <= (others => '0');

   elsif (Clk_i = '1' and Clk_i'event) then

     if (pc_update_l = '1' and Proga_en_i = '1') then   -- when not Monitor mode, update PCL
        pcl_l   <= Proga_i(7 downto 0);
        pch_l   <= Proga_i(15 downto 8);
     end if;
   end if;
end process pc_reg;

-- ---------------------------------------------------------------
-- ---------------------------------------------------------------
--
--  MCU Debugger Logic
--
-- ---------------------------------------------------------------
-- ---------------------------------------------------------------

-- When debugger is not active, global debugger enable and write to stack pointer
-- check whether break point condition is matched
-- directly use the Write_stack_ptr_i and Stack_pointer_i from M8051 to
-- check if the trap condition is met or not

stack_breakpoint: process (debugger_active, bpcrl_bpe_reg_l, Write_stack_ptr_i,
                           Stack_pointer_i, sbk_l, bpcrl_s_reg_l)
begin
  if (((not debugger_active) and bpcrl_bpe_reg_l and Write_stack_ptr_i) = '1' ) then
    if ((Stack_pointer_i > sbk_l and bpcrl_s_reg_l = "11") or
        (Stack_pointer_i = sbk_l and bpcrl_s_reg_l = "01") or
        (Stack_pointer_i < sbk_l and bpcrl_s_reg_l = "10")) then

       stack_trap_l <= '1';
     else
       stack_trap_l <= '0';
     end if;
   else
     stack_trap_l <= '0';
   end if;
end process stack_breakpoint;

-- This logic remembers the trap codition, and it can wait for
-- the last cycle of the current instruction
```

```
ck_regs: process (Clk_i, Resetz_exclude_USB_i)
 in
 f (Resetz_exclude_USB_i = '0') then
  stack_trap_dl    <= '0';

 lsif (Clk_i = '1' and Clk_i'event) then
  if (stack_trap_l = '1') then
    stack_trap_dl  <= '1';
    elsif (prog_mux_control_l = '1') then   -- when in jump to monitor mode, clear stack trap latched signals
      stack_trap_dl <=  '0';
  end if;
 nd if;
  process stack_regs;

  ck_trap_d2 <= stack_trap_l or stack_trap_dl;  -- when stack_trap_l is set, it may be the last cycle

  --------------------------------------------------------------------------------
  trigger jump when last instruction cycle is reached, and set_stack_trap = 1
  --------------------------------------------------------------------------------

  ick_trap_d3 <= Last_cyc_i and State_0_i and (not Internal_wait_i) and stack_trap_d2;

  )cess (Clk_i, Resetz_exclude_USB_i)
 jin
 if (Resetz_exclude_USB_i = '0') then
    stack_trap_d4 <= '0';
 alsif (Clk_i = '1' and Clk_i'event) then
    if (stack_trap_d3 = '1') then   -- extend stack trap time until jump to monitor
      stack_trap_d4 <= '1';
    elsif (prog_mux_control_l = '1') then   -- when in jump to monitor mode, clear stack trap latched signals
      stack_trap_d4 <= '0';  -- clear when debugger is active
    end if;
  end if;
 d process;

 ack_trap_d5 <= stack_trap_d3 or stack_trap_d4;  -- hold stack trap condition until jump_l = 1

  --------------------------------------------------------------------------------
  When stack trap condition occurs, wait for program memory fetch cycle to jump to monitor
  program.
  --------------------------------------------------------------------------------

 .ack_trap_jump_l <= Proga_en_i and stack_trap_d5;

- --------------------------------------------------------------------------------
- address break point logic
-
- When detect stack trap, it will disable address break point function.
- --------------------------------------------------------------------------------

 o1_adr <= bph1 & bpl1;   -- break point 1 address - 16 bits

 p2_adr <= bph2 & bpl2;   -- break point 2 address - 16 bits

 p3_adr <= bph3 & bpl3;   -- break point 3 address - 16 bits

 p4_adr <= bph4 & bpl4;   -- break point 4 address - 16 bits

 reakpoint_logic: process (bank_l, Proga_i, Proga_en_i, bpcrl_bpe_reg_l, stack_trap_d2, debugger_active,
                   bpcrl_be_reg_l, bnk1_ben, bnk2_ben,  bnk3_ben, bnk4_ben, bnk1_b, bnk2_b,
                   bnk3_b, bnk4_b, bp1_adr, bp2_adr, bp3_adr, bp4_adr)
 egin
  bp1_trap_l <= '0';
  bp2_trap_l <= '0';
  bp3_trap_l <= '0';
  bp4_trap_l <= '0';

  -- when blobal debugger enable and not stack trap and not debugger_active and program memory fetch cycle

  if ((bpcrl_bpe_reg_l and (not (stack_trap_d2 or debugger_active)) and Proga_en_i) = '1') then

    if (bpcrl_be_reg_l(1) = '1' and (bnk1_b = bank_l or bnk1_ben = '0') and Proga_i = bp1_adr) then
      bp1_trap_l <= '1';

    elsif (bpcrl_be_reg_l(2) = '1' and (bnk2_b = bank_l or bnk2_ben = '0') and Proga_i = bp2_adr) then
      bp2_trap_l <= '1';

    elsif (bpcrl_be_reg_l(3) = '1' and (bnk3_b = bank_l or bnk3_ben = '0') and Proga_i = bp3_adr) then
      bp3_trap_l <= '1';

    elsif (bpcrl_be_reg_l(4) = '1' and (bnk4_b = bank_l or bnk4_ben = '0') and Proga_i = bp4_adr) then
      bp4_trap_l <= '1';

    end if;
  end if;
 end process breakpoint_logic;

-- --------------------------------------------------------------------------------
-- bpx_trap_active -> address break point condition is active, and will cause jump tom monitor
-- program.
--
-- bpx_trap_active is active when break point address is matching and it occur on odd number.
-- address break point occurs on even number when monitor return to break point address and it
```

```
   should not trigger long jump to monitor program.
-----------------------------------------------------------------------------------------

  1_trap_active <= bp1_trap_l and (not bp1_toggle_l);

  )2_trap_active <= bp2_trap_l and (not bp2_toggle_l);

  )3_trap_active <= bp3_trap_l and (not bp3_toggle_l);

  )4_trap_active <= bp4_trap_l and (not bp4_toggle_l);

  ump_ready <= stack_trap_jump_l or bp1_trap_active or bp2_trap_active or bp3_trap_active or bp4_trap_active;

  ump_gen: process (Clk_i, Resetz_exclude_USB_i)
  egin
   if (Resetz_exclude_USB_i = '0') then
     prog_mux_control_l  <=  '0';

   elsif (Clk_i = '1' and Clk_i'event) then

     if (clear_jump_l) then  -- clear_jump_l is boolean
       prog_mux_control_l  <=  '0';
     elsif (jump_ready = '1') then
       prog_mux_control_l  <=  '1';
     end if;
   end if;
  end process jump_gen;

  jump_proga_en <= prog_mux_control_l and Proga_en_i;  -- jump to monitor and program memory fetch enable

  process (Clk_i, Resetz_exclude_USB_i)
  begin
   if (Resetz_exclude_USB_i = '0') then
     jump_adr_sel <= "00";

   elsif (Clk_i = '1' and Clk_i'event) then

     if (jump_proga_en = '1') then  -- when jump and fetch instruction
       if (clear_jump_l) then
         jump_adr_sel <= "00";  -- when clear jump, clear jump address select
       else
         jump_adr_sel <= (jump_adr_sel + 1);  -- other increment jump address select by 1
       end if;
     end if;
   end if;
  end process;

  jump_logic: process (jump_adr_sel, jtmh_l, jtml_l)
  begin
   case jump_adr_sel is
     when "00"   => Jump_progdi_o  <= "00000010"; --02H, M8051 long jump instruction
     when "01"   => Jump_progdi_o  <= jtmh_l;     -- jump to monitor address reg. (HB)
     when "10"   => Jump_progdi_o  <= jtml_l;     -- jump to monitor address reg. (LB)
     when others => Jump_progdi_o  <= jtml_l;
   end case;
  end process jump_logic;

-- ----------------------------------------------------------------------------------------
-- When in jump to monitor mode and program memory fetch and jump address select = 10, clear jump
-- to monitor mode (clear prog_mux_control_l and stack trap latched signals
-- ----------------------------------------------------------------------------------------

  clear_jump_l <= (jump_proga_en = '1' and jump_adr_sel = "10");

  toggle_reg: process (Clk_i, Resetz_exclude_USB_i)
  begin
   if (Resetz_exclude_USB_i = '0') then
     bp1_toggle_l   <=  '0';
     bp2_toggle_l   <=  '0';
     bp3_toggle_l   <=  '0';
     bp4_toggle_l   <=  '0';

   elsif (Clk_i = '1' and Clk_i'event) then

     if (bp1_trap_l = '1') then
       bp1_toggle_l    <=  not bp1_toggle_l;
     end if;

     if (bp2_trap_l = '1') then
       bp2_toggle_l    <=  not bp2_toggle_l;
     end if;

     if (bp3_trap_l = '1') then
       bp3_toggle_l    <=  not bp3_toggle_l;
     end if;

     if (bp4_trap_l = '1') then
       bp4_toggle_l    <=  not bp4_toggle_l;
     end if;
   end if;
  end process toggle_reg;
```